SOCIAL NETWORK

(Analysis and Web-based Entity Search)

Document Version Control			
Version	**Updated on**	**Change particulars**	**Updated by**
1.0	Aug 02, 2021	Document created	Vishva Shah

Table of Contents

Objective: This document enlists detailed information about the features of the product, high level design decisions, plan of implementation and future scope of the project.

About the product: This system has been designed to

i. Cleanse, consolidate and model the dataset from text files and load it onto a graph database
ii. Provide a web interface to -
   - search entities by any combination of label, properties and their values, with option of exact match or fuzzy match for values of entity's attributes
   - identify cliques that exist in the social network graph

Technology stack overview:

Host language: Python v3.8.0

Database (cloud service): Neo4j Aura

Web framework: Flask v

Front-end: HTML, CSS and JavaScript

Virtual Server: AWS EC2 (Ubuntu Server 20.04)

Virtual consultations:

Date	With	Agenda	Duration
July 27, 2021	Mr. Nikhil Almeida	Introduce the challenge and discuss the problem	15 minutes
July 30, 2021	Mr. Nikhil Almeida	Share progress update and discuss deliverables	15 minutes

Part 1: MODEL THE DATA

After understanding the data provided, I based my decision to use a graph database on the following points:

i. The data had multiple entities, of multiple categories (or type)
ii. Entities have some or all of the attribute values that it can take under their type, which are distinct
iii. Entities are related to other entities by relationships
iv. Analysis on the network requires querying on the relationships

Graph database is perfectly suited for such a use case where insights about the map, connected components and relationships amongst entities are important.

Overview:

Code file name: socialNetwork.py

In this code, I have used the software library "pandas" to read the dataset from the text files as csv, while removing the leading and trailing spaces from some data cells.

Using the Neo4j Python Driver 4.3 to connect to the cloud-based graph database instance on Neo4j Aura, all data was loaded onto the database instance.

Sequence of steps followed:

- Created an account on Neo4j Aura and created a blank instance.
- Stored the connection URL, username and password for the instance as environment variables NEO4J_DB, NEO4J_USR and NEO4J_PWD respectively
- Installed neo4j python driver 4.3

Code flow and functions:

- The dataset text files are read as CSV using methods in pandas by specifying the delimiter (in this case, tab ie "\t") with the argument skipintialspace set to True, which trims off the blank space at the beginning of a cell value
- This dataset is passed to the function trim(), where I have defined a lambda function to apply strip to all string values in the dataset, thereby removing trailing blank spaces, if any in the data values
- Connected to the cloud database and opened a new session
- First loaded the entity-properties dataset onto the database
    - Considering 'ID' unique, all entries were made with the MERGE query in cypher, to prevent creating duplicate nodes while adding multiple attributes for an entity
- Next, created the entity-relationships between entities, identified by ID
- Closed the session and driver

Part 2: WEB BASED ENTITY SEARCH AND ANALYSIS

Overview: This web app is created on the Flask framework and deployed on the standalone WSGI server Gunicorn.

PHASE: <u>DEVELOPMENT</u>

Preparing the system for flask:

Set up python, pip and virtualenv.

Installed python dev-tools (development tools)

Installed pip – package manager that allows us to install other application packages

Installed virtualenv – creating a virtual environment contains the python app with all its dependencies within its own isolated environment

Application directory structure:

-app.py	the main python app, where all functions and redirections are defined
-static	stores static content for the webapp
-templates	stores the views ie. html pages
-----search.html	landing page of our web app
-test.py	unit tests
-wsgi.py	run app with wsgi
-requirements.txt	lists all packages

requirements.txt

```
click==8.0.1
Flask==2.0.1
gunicorn==20.1.0
Jinja2==3.0.1
MarkupSafe==2.0.1
neo4j==4.3.3
networkx==2.6.2
pytz==2021.1
unittest2==1.1.0
Werkzeug==2.0.1
```

Code flow and function calls:
File: app.py

- index(): Function to redirect landing page url to the '/search' route
- search(): function is called when '/search' route is hit. This function evaluates the http requests and computes & returns results to the frontend
  - gets the parameters from the http get request
  - calls multiple functions within it, to fetch various arguments to pass on to the frontend
  - these functions query the graph database to retrieve all data that will then be passed on to jinja2, to manipulate and display results on the frontend

FEATURES ON THE WEB APP:

1. NETWORK STRUCTURE OVERVIEW
   Overview: This section provides a visual overview of the entities and the network structure
   Implementation:
   Two charts have been rendered using the JavaScript charting library 'AnyChart'
   a. A network graph, representing the entities and connected components
   b. A pie diagram, representing the number of entities of each type
   Opportunity to improve/scale:
   - This section can be made into an interactive visualization, where the user can click on the nodes to display its attributes, highlight nodes belonging to same type, highlighting its relationship type with connected components.

- For larger datasets, we can have collapsible network graphs here, to collapse and expand certain subsets or groups of the graph and be able to see the high-level structure or drill down to a very specific sub-section of the graph.

2. ENTITY SEARCH

Overview: This section allows entities to be searched through the database by any combination of one or more of these – label, property, value, with option to allow fuzzy matches for attribute values

Implementation:

User can submit any combination of parameters from the below:

- Choosing a label from the dropdown where all distinct labels from the data are listed
- Choosing an attribute from the dropdown where all distinct properties form the data are listed
- Type in text to match by the attribute value

  *by default the text match is exact. To allow fuzzy matches, checkbox fuzzy match should be checked

Search parameters put in by the user are passed as url parameters upon form submission
The getResults function in the main app file has a series of conditional statements to retrieve results based on the combination of search parameters:
All 7 possible cases of params combination are handled:

i. Only label: All nodes which have the said label are fetched
ii. Only value: All nodes where any attribute has the said value are fetched
iii. Only property: All nodes that possess the said property are returned
iv. Label and property: All nodes having the said label and having the said property are returned
v. Label and value: All nodes having the said label and any property having the said value are returned
vi. Property and value: All nodes having the said value for the said property are returned
vii. Label, property and value: All nodes having the said label and an attribute-value pair, matching the said property and value are returned.

Additionally, the database is queried to retrieve approximate searches based on attribute values if fuzzy match option is checked

Opportunity to improve/scale:

- User can be allowed to select the threshold for fuzzy match.
  For example, depending on their needs and constraints, one user may consider a 70% string match as a valid match while another user may strictly want entities to be found matching only if more than 90% of the text string matches.

- For larger datasets of this type, this feature can be extended to perform record linkage and entity resolution, where a lot of duplicate entities with slight inconsistencies may be present.

3. CLIQUES
   Overview: In this section, one can view the subsets of entities that are identified as graph cliques in the data
   Implementation:
   - Information about all nodes and connected components is retrieved by querying the database.
   - Edges are stored as txt, identified by the IDs of the two connected entities.
   - Using the function in networkx library, return the maximal cliques present in the data as a list of lists

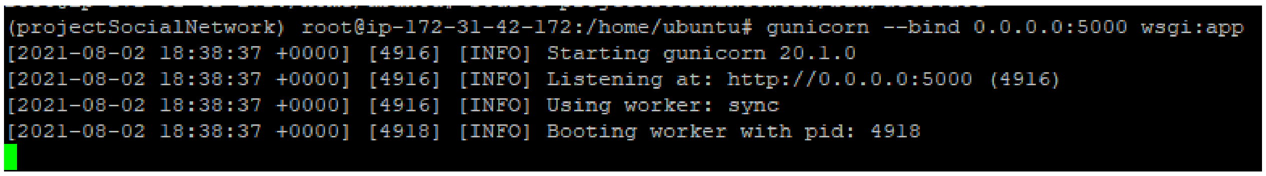
   Opportunity to improve/scale:
   - For larger datasets of this type, one may want to allow selecting the minimum number of entities in a clique.
     For example, for very large datasets with large number of relationships, information about a clique of just 3 entities may not be very insightful.
   - For larger datasets, we may calculate cliques on a particular group/label or relationship type.
     For example, to find a close group of friends, we may only consider nodes connected by the 'FRIENDS_WITH' relationship
     Or we may only want to find cliques between entities of the type 'Person' to identify group of persons strongly connected by different relationships

PHASE: DEPLOYMENT

The flask app has been deployed on the standalone WSGI server Gunicorn.

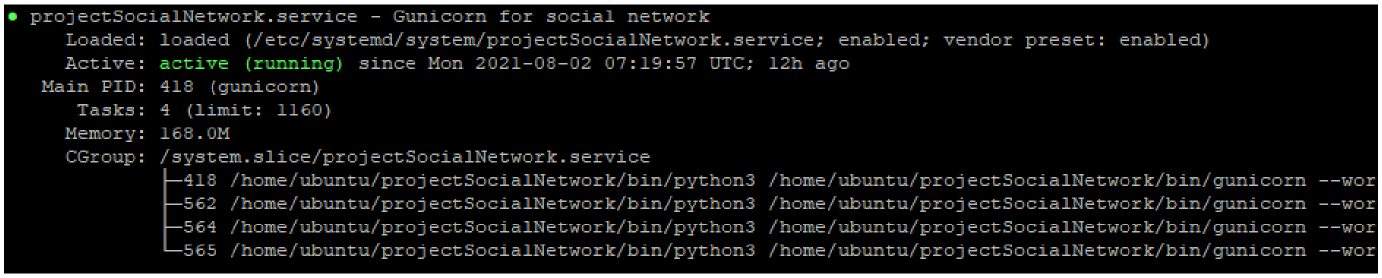
Steps followed:

- install gunicorn 20.1.0 with pip
- created the wsgi file and set it as the entry point for the python app
- created a separate screen session on ubuntu where application will be served continuously
  terminal command: screen -S appSocialNetwork
- bind gunicorn to the wsgi file so that the application will run with the wsgi file
  terminal command: gunicorn –bind 0.0.0.0:5000 wsgi:app



```
(projectSocialNetwork) root@ip-172-31-42-172:/home/ubuntu# gunicorn --bind 0.0.0.0:5000 wsgi:app
[2021-08-02 18:38:37 +0000] [4916] [INFO] Starting gunicorn 20.1.0
[2021-08-02 18:38:37 +0000] [4916] [INFO] Listening at: http://0.0.0.0:5000 (4916)
[2021-08-02 18:38:37 +0000] [4916] [INFO] Using worker: sync
[2021-08-02 18:38:37 +0000] [4918] [INFO] Booting worker with pid: 4918
```

- create the service file for gunicorn at the location /etc/systemd/system/projectSocialNetwork.service
- start the service file using the command systemctl start projectSocialNetwork; It creates a socket file and then we enable service using the command systemctl enable projectSocialNetwork
- once the service is enabled, we can check its status by running the command systemctl status projectSocialNetwork



```
● projectSocialNetwork.service - Gunicorn for social network
     Loaded: loaded (/etc/systemd/system/projectSocialNetwork.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2021-08-02 07:19:57 UTC; 12h ago
   Main PID: 418 (gunicorn)
      Tasks: 4 (limit: 1160)
     Memory: 168.0M
     CGroup: /system.slice/projectSocialNetwork.service
             ├─418 /home/ubuntu/projectSocialNetwork/bin/python3 /home/ubuntu/projectSocialNetwork/bin/gunicorn --wor
             ├─562 /home/ubuntu/projectSocialNetwork/bin/python3 /home/ubuntu/projectSocialNetwork/bin/gunicorn --wor
             ├─564 /home/ubuntu/projectSocialNetwork/bin/python3 /home/ubuntu/projectSocialNetwork/bin/gunicorn --wor
             └─565 /home/ubuntu/projectSocialNetwork/bin/python3 /home/ubuntu/projectSocialNetwork/bin/gunicorn --wor
```

Opportunity to improve:

- Auto-reload: We can enable auto-reload with gunicorn to automatically update code changes without having to restart the server.
- We can set reverse proxy to our registered domain in nginx config file,
  where we configure the server name to our domain, proxy parameters (socket file path)
  Enable the site and restart nginx server and the site should be up and running on the custom domain

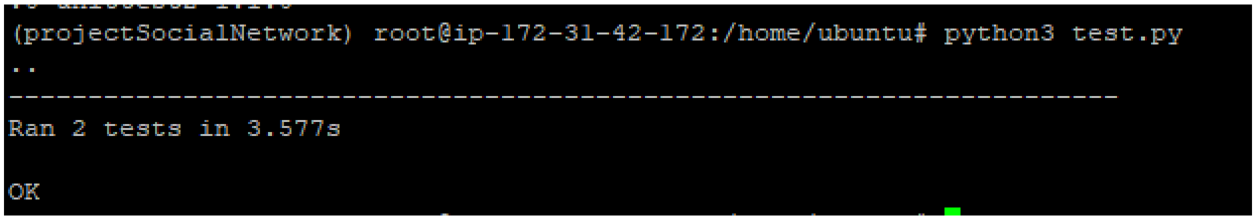
TESTING THE WEB APP

I have written unit tests in the tests.py file in the project directory, using the unittest2==1.1.0 library in python.

1. The first unit test is to assert the status code upon hitting the url ('/')
   As per our code, this should get redirected to ('/search'), hence the status code should be 302.
   *Status code 302 indicates that route was found, and redirected to another location

2. The second unit test is to assert the status code upon hitting the url ('search')
   This should respond with the status code 200.
   *Status code 200 indicates success -request completed

On running the file tests.py, following output shows on the terminal:



```
(projectSocialNetwork) root@ip-172-31-42-172:/home/ubuntu# python3 test.py
..
----------------------------------------------------------------------
Ran 2 tests in 3.577s

OK
```

Opportunity:

- We can perform additional tests to check the parameters passed through http get request, check content loaded on the page against the parameters
- We can develop a test suite to create stubs on mock data and test database operations.

ACCESSING THE WEB APP

To access the web app, click here
Or type this url address in your browser “http://13.58.210.215:5000/search”

LINK TO SOURCE CODE

All source code, static files and requirements.txt added to this github repo.
Or type this url address in your browser “https://github.com/vishva1001/socialNetwork”